

2 ANALYSIS

(COMMON)

2.1 VMWARE SNAPSHOTS

A snapshot is used to save a specific state of the virtual machine in order to come back to this state later in the time. To allow that VMware has to copy and modify some virtual machine files. Now we will succinctly explain what VMware really does with the virtual machine when it takes a snapshot.

Take the case that you are running a virtual machine which is composed by two files:

- VMDK this file is the image of the virtual machine file system
- VMX this file is the descriptor of the virtual machine

You can make a snapshot when the virtual machine is running or powered off. Let's first talk about what happens when the virtual machine is powered off. The only thing that you need to come back to the snapshot is to know all the changes that occurred after that snapshot. VMware realizes this by creating a new VMDK file. This new file will contain all the modifications done by the user after the snapshot was taken. So it is quite simple to restore a snapshot: VMware has just to delete the new VMDK and modify the configuration file to use the old VMDK.

When VMware takes a snapshot of a running virtual machine, the main mechanism is the same except that VMware has also to save the state of the RAM memory. This means that if you take a snapshot of a virtual machine which is running, VMware will produce more than a file.

The main problem we encountered is that the version of VMware we use (VMware Server) allows only one snapshot. This means that if you already have a snapshot of your virtual machine, the next snapshot that you will take will override the previous one.

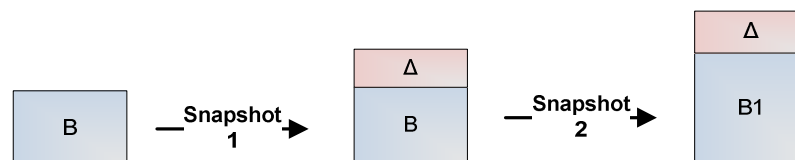


Figure 1 – VMware Server Snapshot Mechanism

The Figure 1 shows the snapshot mechanism of VMware Server. The first blue rectangle represents the base VMDK image file and the red rectangle represents the snapshot difference file (delta). When you take the first snapshot there is no problem, the base VMDK image file is lock in its state and a new VMDK file is created. Now all the change that you apply to the virtual machine will be written into this snapshot file. The problem comes at the next step, when the user makes the second snapshot. VMware makes a merge of the previous snapshot file and the base image file to create a new base image file. This last one contains the first base image file and all the modifications done until when the user took the snapshot.

After the second snapshot, we can only come back to the base image B1. It is impossible to come back to the base image B, because the difference file created during the first snapshot is lost and the base VMDK file has been modified.

3.3 VML MAIN COMPONENTS

The physicist works with virtual machines and wants to save the state of his virtual machines during his work. In VML, the folders which contain the virtual machines files are called *working areas*. When VML saves the state of a virtual machine, actually it copies some files of the VM and stores some metadata into another folder called *repository*. These files and this metadata compose a VML entry.

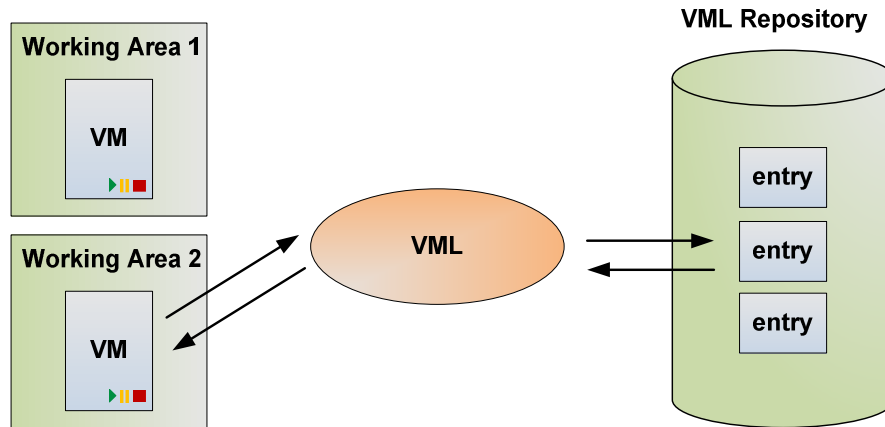


Figure 7 – General Architecture of VML

Figure 7 shows the general architecture of VML with the main component. The virtual machines in the working areas can be directly used by the physicist (i.e., he can start, stop and suspend them). A working area can contain zero or one virtual machine, but the user can have more than one working area.

The entry into the repository corresponds to virtual machines, but the user cannot directly use them from inside the repository. If he wants to open an entry, he must ask VML to do it. VML then extracts the entry into a working area and the physicist can then use the virtual machine.

To resume, the two fundamental processes done by VML are:

- save a virtual machine: VML copies some data from the virtual machine and creates a new entry into the repository;
- build an entry: VML extract the entry from the repository and recreates the virtual machine inside a working area.

3.3.1 VML Entries

The VML repository contains entries, which are descriptions of virtual machines at a given state⁴. One of the VML central component is the VM-entry converter. It provides the conversion in both directions: it can transform a virtual machine in an entry and an entry in a virtual machine.

Basically, VML could contain entire virtual machines instead of “complex” entries. The problem is that a whole virtual machine may need a large amount of disk space. The idea of VML is to reduce this size by not storing redundant information. VML stores only some parts of the virtual

⁴ Here “state” does not refer to the running, paused or shut down state of the VM. With “state of a virtual machine” we refer the state of the VM’s hard disk, the memory state and so on.

machine into the repository. To be able to recreate a virtual machine from these parts, it must also have some metadata which explain how to rebuild the original VM. These “parts of VM” and the metadata compose a VML entry.

VML cannot always store only some parts of a virtual machine. Sometimes it is necessary to store the whole virtual machine into the repository. An entry which contains a whole virtual machine is called a *full entry*.

An entry which doesn’t contain a whole virtual machine is called a *partial entry*. VML can create a partial entry only from a virtual machine which respects a pre condition: the parts of the virtual machine which will not be stored in the repository can be found elsewhere⁵. This pre condition is necessary to make VML able to rebuild the partial entry.

3.3.2 Storage and Restoration of the Entries

VML has different ways to create partial entries. Next sections explain the different techniques used by VML to store full and partial entries and to rebuild the virtual machines from these entries.

3.3.2.1 Incremental Backups

Incremental backups is a technique currently used by the virtualization platform to take snapshots of virtual machines. VML will actually use the snapshots utility to manage the incremental backups. The idea is that at each time the user saves a virtual machine, VML stores only the differences from the last backup. Figure 8 shows an example of incremental entries backups. In the example, the physicist works in a virtual machine and during his work, he wants to save the state of the VM.

At the beginning, the virtual machine is in a state S1 and the physicist executes the “save” command of VML. To be able to restore this virtual machine, VML needs to store a whole copy of the VM into the repository, so the entry E1 which correspond to the VM state S1 contains the entire machine.

The physicist continue his work into the machine and then executes another VML’s “save” command. At this moment the virtual machine is in the state S2. Probably only a few amount of data changed from the state S1, so VML doesn’t copy to the repository the entire VM, but only the differences between states S1 and S2. These differences compose the new entry E2. The disk space occupied by E2 is probably much less than the space occupied by E1.

The physicist works in the machine, which is now in the state S3. When he asks VML to save this state, VML stores into the repository the difference between states S2 and S3 and calls this entry E3. At this moment, the repository contains three entries: E1, E2 and E3. Each entry corresponds to a different virtual machine state. E1 is a full entry and E2 and E3 are partial entries.

⁵ For VML, “elsewhere” is another entry in the repository.

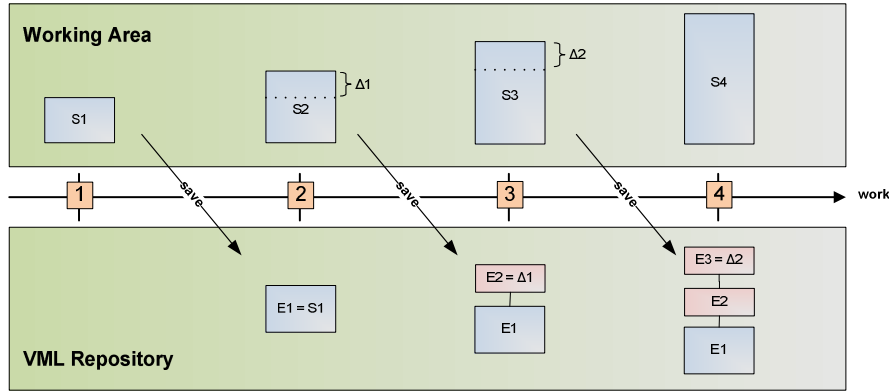


Figure 8 – Incremental Backups in VML

Figure 8 shows how the working area and the repository evolve during the work of the physicist. The part upside the arrow represents the files of the virtual machine placed in the working area. The physicist works in the virtual machine and we represent this by increasing the size of the rectangles. The part of the figure below the arrow is the VML repository. A file in this area means that it is saved into the repository. At each step is represented the state of the working area and the content of the repository.

Let's consider the content of the repository at step 4. If the physicist wants to restore the state S1 in the working area, VML will copy the whole content of E1 to the working area and the machine is restored. However, if the physicist wants to restore the state S2, VML must also copy the whole content of E1 to the working area and then add the content of E2 (i.e. the difference between states S1 and S2). If the physicist asks to restore the state S3, VML must do the same than for the state S2 and then add the content of E3.

3.3.2.2 Differential Backups

Incremental backups is not the only technique which can be used to create partial entries. Let's imagine a physicist who works on a virtual machine and who uses incremental backups of the VM. After a year, he may have tenth of entries in his repository. Before to restore the entry n , VML must restore the entry $n-1$, which needs that the entry $n-2$ has been restored... and so on to the entry 1. Restore an entry may then take too much time.

A solution of this problem comes with the differential backups. With this technique, all the partial entries refer to only one entry which is stored into the repository. We call this entry the *base*. When VML must build an entry, it has just to restore the base entry and the partial one.

Figure 9 is the equivalent of Figure 8, but in this case VML uses differential backups. The entry E1 is the base image and E2 and E3 are the partial entries. We see the difference with the incremental backup at the step 3. Instead of taking a difference between S2 and S3, VML stores the difference between S1 and S3 into the entry E3.

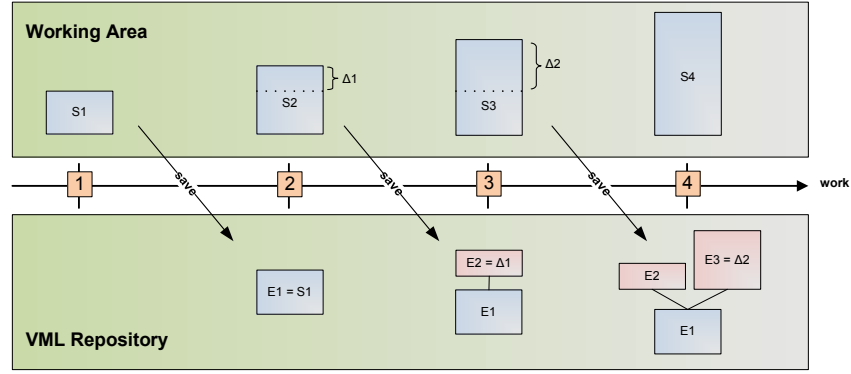


Figure 9 – Differential Backups in VML

3.3.3 Dependencies Between Entries

Let's consider the three entries created during the example in Figure 8. E1 was the full entry and E2 and E3 were two partial entries taken with the incremental backups. To restore E2, VML needs E1. To restore E3, VML needs E2 and E1. We say that E3 *depends* on E1 and E2. E2 also depends on E1. Note that an entry can depend on more than one (like E3), but an entry depends *directly* always from only one entry (or zero)⁶. When entry E2 directly depends on entry E1, we say that E1 is the *parent* of E2. In Figure 8 and in Figure 9 we show this dependency with a connection line between the two entries.

We can represent the dependencies between entries with trees: at the bottom we have the parent entries and at the top we have the partial entries which depends on the parents. The content of the VML repository can be represented with a collection of trees, as show in Figure 10.

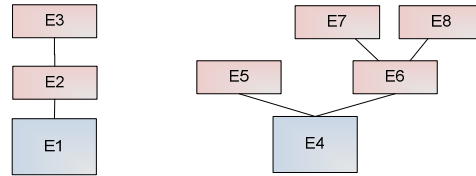


Figure 10 – Tree Structure of the Entries in the VML Repository

The tree on the left side of Figure 10 is generated by a sequence of incremental snapshots. The tree on the right side is more complex: entries E5 and E6 both directly depends on E4. Entries E7 and E8 depends on E6. The tree on the right could have been generated by the following sequence of actions with incremental backups:

- Save twice the state of the VM. The first time VML creates the full entry E4 and then the partial entry E5.
- Restore the state of the entry E4.
- Save twice the state of the VM. The first created entry (E6) directly depends on E4. The second entry (E7) depends on E6.
- Restore the state of the entry E6.

⁶ This is true while we don't have two entries which are exactly the same into a repository. Even in this special case, we can simplify and consider that a third entry only depends on the first or the second of the two twin entries.

- Save the state of the VM. VML creates the entry E8.

This is not the only way to generate the tree on the right. We can combine differential and incremental backups, like in this sequence of actions:

- Save the state of the VM. VML creates the full entry E4. This is the base for all the differential snapshots.
- Save twice the state of the VM with differential backups. VML creates the two entries E5 and E6.
- Save the state of the VM with an incremental backup. The difference between the current state of the VM (composed by E4+E6+E7) and the last one (E4+E6) is E7.
- Restore the state of the entry E6.
- Save the state of the VM. VML creates the entry E8.